International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5
***Article Received:** 25 March 2023 **Revised:** 12 April 2023 **Accepted:** 30 May 2023*

AI-Driven Predictive Caching for Edge Computing

Ashlesha Gupta

Assistant Professor, Department of Computer Engineering, J.C. Bose University of Science and Technology, YMCA, Faridabad, India,

gupta_ashlesha@yahoo.co.in

Abstract

The exponential growth of digital content consumption is straining traditional centralized networks, resulting in high latency and server overloads. To meet user expectations, Edge Computing has emerged as a vital solution, placing processing and storage resources near end-users. Within this decentralized architecture, caching is critical for efficient content delivery. However, conventional caching methods like LRU and LFU are static and fail to adapt to dynamic user behavior and evolving content popularity. This paper proposes an AI-Driven Predictive Caching framework to overcome these limitations. The approach utilizes the Random Forest Algorithm to analyze historical access patterns and proactively forecast future content demands. This predictive strategy is demonstrated using Redis to simulate a realistic, in-memory edge environment, with Streamlit providing real-time visualization of system behavior and predictions. Our findings aim to demonstrate that integrating machine learning with edge computing significantly enhances traditional caching mechanisms, resulting in higher cache hit rates, reduced latency, and ultimately creating more adaptive and responsive edge networks essential for critical content delivery systems

Keywords: Edge Computing, Predictive Caching, Artificial Intelligence (AI), Random Forest Algorithm, Cache Hit Rate

1. Introduction

As the use of Internet and use of digital content has increased in recent years, there is a need for quicker and more efficient methods of delivering content. This increase in usage has placed pressure on traditional centralized networks. These networks fails to meet user expectations due to factors like latency, bandwidth limitations, and server overloads. To cater to these challenges, use of edge computing is gaining importance. Edge computing poses to be an effective approach that places processing and storage resources near to the end user. This proximity to the end users helps in reducing data transmission delays, saving bandwidth, and improving the overall user experience [1].

Caching serves as an important tool for edge computing systems to function effectively, in content delivery scenarios. By retaining copies of frequently requested content nearer to the users, caching promotes faster response time and reduces the need of fetching data from centralized servers. Many Conventional caching techniques like Least Recently Used (LRU) and Least Frequently Used (LFU) have been excessively used due to their straightforward implementation. However, these rule-based methods suffer from problems like inability to shifts based on user behavior, evolving content popularity, and changing network conditions [5][6].

Predictive caching driven by Artificial Intelligence (AI) has been thus proposed in this paper to overcome the above discussed limitations. The proposed method works by analyzing historical access patterns and forecast future content requests based on Random Forest Algorithm machine learning models. As the proposed approach is based on proactive caching strategy, it is able to store relevant user content thereby increasing cache hit rates, reducing latency, thereby enhancing the performance and responsiveness of edge computing networks.

In the proposed method, Redis has been used to simulate a realistic edge environment in-memory caching system. For visualizing predictions and system behavior in real-time Streamlit has been used. The proposed method aims to demonstrate that AI can effectively enhance traditional caching mechanisms and make edge networks more adaptive and responsive. The combination of machine learning with edge computing has the potential to significantly improve content delivery systems, particularly in scenarios where network performance is critical [7].

2. Literature Review

In edge environments, low-latency delivery is critical. Here, caching helps reduce the need to fetch data from central servers, saving bandwidth and reducing response time. However, the constraints of limited storage, computing power, and constantly changing user demands make traditional caching methods suboptimal. AI-based approaches introduce learning capabilities into caching, allowing the system to make context-aware, adaptive decisions.[1][5]. AI enhances caching performance by Learning access patterns over time instead of fixed rules, adapting to new data trends (e.g., shifting user behavior, location-specific content needs), incorporating multiple features (like device type, time of day, content popularity) and predicting future requests, making cache hit ratios higher and low latency. Machine Learning is a subset of AI which empowers systems to identify patterns within data and improve their decision-making capabilities over time without depending on hard coded instructions [8][32][33]. When applied to caching systems, ML techniques are instrumental in forecasting future content requests by analyzing user activity and relevant system metrics. Machine learning techniques introduce data-driven intelligence into caching mechanisms, enabling systems to:

Identify and analyze user access patterns,

Extract insights from temporal attributes, such as time of request,

Adapt dynamically to spatial factors like device type and user location,

Predict future content demands, thereby optimizing cache storage by prefetching or retaining frequently accessed or high-value data

Table 1 provides an overview of the Machine Learning Algorithms for Caching Optimization.

ML Algorithm	**Expected Accuracy**	**Resource Usage**	**Model Transparency**	**Suitability for Real-Time Use**
Random Forest	Reliable	Moderate	Fairly Interpretable	Reasonably Suitable
Support Vector Machine	Strong	High	Limited Transparency	Less Suitable
Neural Networks	Very Strong	Resource Intensive	Low Interpretability	Moderately Feasible
K-Nearest Neighbors	Average	High	Easy to Interpret	Not Ideal
Decision Tree	Moderate	Lightweight	Highly Transparent	Well-Suited
Gradient Boosting	High	High	Low Transparency	Moderate

Over the last decade, numerous studies have addressed intelligent caching systems using machine learning, especially within the contexts of web services, mobile networks, and, more recently, IoT environments. Table 1.2 summarizes and compares representative works to identify their contributions, strengths, and shortcomings [9][34][35].

Table .2: Comparative Analysis of Existing Works

Study	**Approach**	**Model Used**	**Domain**	**Contribution**
Goudarzi et. al. [1]	**Proactive caching using ML**	**Random Forest**	**Mobile Edge**	**Predicts user content requests based on mobility patterns**

***Article Received:** 25 March 2023 **Revised:** 12 April 2023 **Accepted:** 30 May 2023*

Keshari et. al., [4]	Edge caching with federated learning	LSTM + FL	IoT	Privacy-preserving caching via decentralized training
Gao et. al. [2]	Context-aware caching	XGBoost	Smart Cities	Uses context (location, time, device) to improve hit rate
Alkadi et. al., [3]	Lightweight ML caching	TinyML / SVM	Embedded IoT	Focused on reducing model size and energy for edge devices

A deep analysis of latest literature on intelligent caching mechanisms, particularly those which have Machine Learning (ML), reveals some limitations that hinder real-world deployment—especially in resource-constrained environments like IoT or edge computing systems:

Lack of Edge Adaptability:

Many existing models are trained on cloud infrastructure, making them hard to deploy on low-power edge devices because of size, complexity, and runtime requirements.[9] [10][36][37]

Heavyweight and Complex Models:

LSTM, GRU, or CNN, while accurate, are often computationally expensive solutions. They have latency and energy overheads, limiting their applicability in latency-sensitive environments.[11]

Static and Non-Adaptive Caching:

Traditional caching mechanisms fail to dynamically adapt to non-uniform user behavior or environmental changes. ML models that are not regularly updated lead to outdated predictions.[6]

Insufficient Focus on Energy and Latency:

Although important for IoT and smart device applications, only a few of studies provide evaluation metrics such as energy consumption, inference latency, or memory footprint [12]

Dataset Limitations:

Most publicly available datasets lack contextual information relevant for caching, such as user mobility, content freshness, and edge deployment metadata.[10]

Limited Comparisons Across Models:

Several studies introduce individual machine learning models without conducting comparative benchmarks against alternatives such as Random Forest, XGBoost, or Neural Networks, often overlooking the trade-offs between model performance and computational complexity[6]

The proposed work aims to design a predictive caching framework that is:

Lightweight and Scalable: Best for real-time deployment on edge devices with limited computational resources.

Comparative in Nature: Evaluates multiple ML models—such as Random Forest, XGBoost, and Neural Networks—to highlight trade-offs in accuracy, inference time, and resource usage.

Real-Time Adaptive: Continuously learns from new data to update caching decisions, reducing cache misses and latency.

Energy and Performance Aware: Uses performance metrics that reflect real-world operational efficiency, especially for smart and low-power devices.

The proposed work aims to bridge the gap between accuracy-focused research and deployment-ready solutions [13][38][39].

3. Proposed Work

The proposed AI Driven Predictive technique for Edge computing is designed to provide intelligent content caching within edge computing scenarios. The main goal of the proposed technique is to build and evaluate a supervised machine learning model that can make real-time caching decisions by analyzing attributes of incoming web requests [14][40].

The proposed methodology follows a series of iterative stages aimed at developing an efficient predictive caching system:

Synthetic Data Creation: Generating simulated user request patterns to form a realistic and representative dataset.

Data Preparation and Feature Development: Improving the dataset through preprocessing steps and systematically designing input features to enhance model learning.

Model Building and Evaluation: Training and assessing multiple supervised learning models to find the best model for predictive caching.

System Integration: Implementing the trained model within a Redis

in-memory caching framework and making a user-friendly interface using Streamlit.

To improve system accuracy and computational efficiency, each of the above-mentioned stages were tested repeatedly and refined.

3.1 Database Design

A synthetic database was first created to simulate the request behavior in content delivery systems. The database was created with the following objectives [41][42][43]

To Represent temporal access patterns (e.g., peak hours, weekends)

To Simulate diverse device and network types,

To incorporate varying cache-worthiness levels for different requests,

To Maintain balance between cache and no-cache labels for classification.

Features Collected

Each data record corresponds to an individual content request, characterized by the following features [44][45]:

Table 3: Features Description

Feature	**Description**	**Type**
user_id	Unique identifier for the user	Categorical
url	Requested content/resource	Categorical
request_count	number of same request by same user	Numerical
response_size	Size of content returned (in KB)	Numerical
time_taken	Response latency (in milliseconds)	Numerical
device_type	Device used (mobile, desktop, tablet)	Categorical
region	Geographical region (e.g., Asia, Europe, US-East)	Categorical
request_meth od	HTTP method (GET, POST, PUT, DELETE)	Categorical
hour	Hour of the day when request was made	Numerical

day of week	The day when the request was made	Numerical
cache label	Binary	Binary Label

Additionally, some tags to enhance context awareness were also created like

is_peak_hour (boolean): True if access occurred during peak usage windows (e.g., 6 PM – 9 PM),
is_weekend (boolean): True for Saturday or Sunday accesses.

3.3 Data Creation

The Python libraries such as NumPy and Faker were used to create the dataset. The data generation process incorporated several strategies [14][15][16]:

To simulate realistic network behaviour latency variations were based on geographic region and device type
To reduce class imbalances, the cache_label classes were balanced to maintain an even 50-50 split.
To simulate real user behaviour some randomness was inserted in data.

The resulting dataset comprised of 20,000 records. The dataset thus created offered ample diversity for training and validation and at the same time lightweight enough for edge-scenario modeling [17][18].

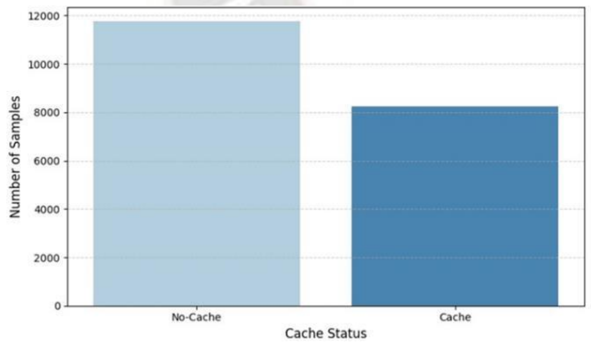


Figure1: Sample Distribution of Cache vs. No-Cache Labels

Data Preprocessing

Firstly to convert the raw request logs into a structured format appropriate for training supervised learning algorithms , they were passed through a systematic preprocessing pipeline. To adjust Numerical features such as response_size and time_taken within a range[0,1], they were normalized using min max scaling [19][20] . The normalization is done avoids bias toward high-magnitude variables and improves training stability. Categorical fields were also encoded using the following strategies:

To generate separate binary columns for each distinct category, One-Hot Encoding was applied to features such as device_type, region, and request_method,

for high-cardinality fields like url and user_id, , Frequency Encoding was used to replace each unique value by its corresponding frequency of occurrence within the dataset [21][22].

This process helped to treat non-numeric attributes as learnable features. Additional context features were derived to capture temporal and behavioral nuances:

is_peak_hour: Flagged requests made between 18:00–21:00 as True.

is_weekend: Set to True for requests made on Saturday or Sunday.

user_activity_level: Computed as the total number of requests made by a user during the week (high/medium/low tiers).

These features enriched the dataset with higher-level semantics. Since the primary label (cache_label) was binary, class balance was maintained by design with a 50:50 distribution [46][47][48]:

Oversampling (SMOTE) and undersampling were tested but not needed due to the controlled generation process.

The resulting well-prepared, feature-enriched dataset was then utilized as the foundation for training the predictive models.

3. 5 Model Selection

Random Forest model was selected due to following reasons:

It has Balanced Accuracy & Interpretability: Outperformed Logistic Regression and Decision Trees on validation sets.

It can Handle synthetic variations in the dataset effectively.

As it is Less intensive than boosting models, it is better suited for eventual edge deployment.

```
pip install  pandas scikit-learn  xgboost  matplotlib  s
```

It has Support for Feature Importance Analysis3.6 Training and Evaluation

A database was split into 80:20 ratio. Model was trained using hyperparameter tuning based on a grid search strategy, a n d evaluating using a different combinations of key parameters such as estimators, the max depth, and the minimum sample size to split a node. To ensure a complete assessment of the model’s performance, some evaluation metrics were used, precision, accuracy, recall, and F1 score. Additionally, a confusion matrix was analyzed to gain deeper insights into classification outcomes [23][24].

```
import pandas as pd1

batch1 = pd1.read_csv("batch1.csv")

batch2 = pd1.read_csv("batch2.csv")

batch3 = pd1.read_csv("gpt_batch7.csv")

batch4 = pd1.read_csv("gpt_batch8.csv")

# Combine all batches

full_data = pd1.concat([batch1,batch2,batch3, batch4])
```

3.7 System Integration

The trained Random Forest model was integrated with a Redis server to simulate real-time caching decisions. A Streamlit application was made for:

Generating sample requests
Displaying model predictions
Showing Redis cache updates

This allowed simulation of edge behavior where decisions are made in real time based on user input [25].

3.8 Deployment Simulation

While actual edge deployment was not conducted, the simulation mimicked a lightweight edge node using:

Local system resources only

Minimal dependencies (Scikit-learn, Redis, Streamlit)

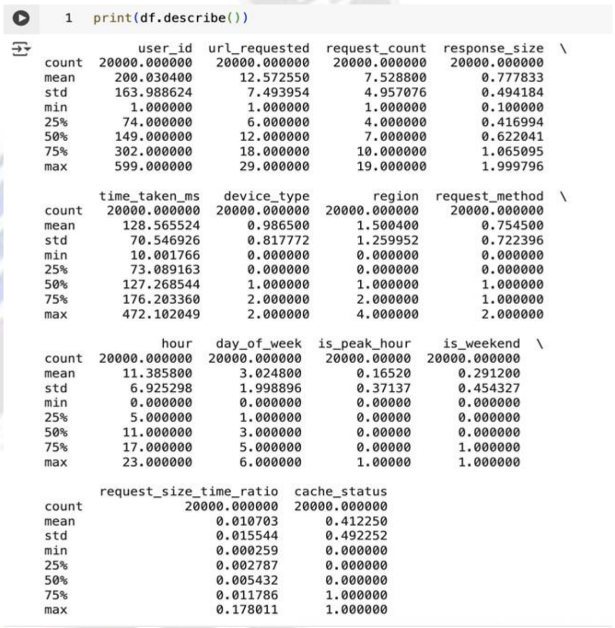
In-memory decision-making via Redis

This setup reflects what would run on real-world edge devices like Raspberry Pi, IoT gateways, or CDN nodes [26].

4. Implementation and Results

Implementation and testing were done mainly in a cloud-hosted environment—Google Colab—which provides scalable computing resources, including GPU acceleration, to support efficient experimentation [27]. All necessary software libraries were managed within the Google Colab environment, utilizing either pre-installed packages or installations through the pip package manager. For local system execution, the required dependencies were installed using the following command:

The system utilized synthetic or preprocessed caching logs stored in CSV format. Four different batches batch1.csv, batch2.csv, gpt_batch7.csv, and gpt_batch8.csv—were concatenated to create a unified dataset named caching_dataset_20k.csv.



```
print(df.describe())
```

```
            user_id  url_requested  request_count  response_size  \
count  20000.000000   20000.000000   20000.000000   20000.000000
mean     200.030400      12.572550       7.528800       0.777833
std      163.988624       7.493954       4.957076       0.494184
min        1.000000       1.000000       1.000000       0.100000
25%       74.000000       6.000000       4.000000       0.416994
50%      149.000000      12.000000       7.000000       0.622041
75%      302.000000      18.000000      10.000000       1.065095
max      599.000000      29.000000      19.000000       1.999796

       time_taken_ms   device_type        region  request_method  \
count   20000.000000  20000.000000  20000.000000    20000.000000
mean      128.565524      0.986500      1.500400        0.754500
std        70.546926      0.817772      1.259952        0.722396
min        10.001766      0.000000      0.000000        0.000000
25%        73.089163      0.000000      0.000000        0.000000
50%       127.268544      1.000000      1.000000        1.000000
75%       176.203360      2.000000      2.000000        1.000000
max       472.102049      2.000000      4.000000        2.000000

               hour   day_of_week  is_peak_hour    is_weekend  \
count  20000.000000  20000.000000   20000.00000  20000.000000
mean      11.385800      3.024800       0.16520      0.291200
std        6.925298      1.998896       0.37137      0.454327
min        0.000000      0.000000       0.00000      0.000000
25%        5.000000      1.000000       0.00000      0.000000
50%       11.000000      3.000000       0.00000      0.000000
75%       17.000000      5.000000       0.00000      1.000000
max       23.000000      6.000000       1.00000      1.000000

       request_size_time_ratio  cache_status
count             20000.000000  20000.000000
mean                  0.010703      0.412250
std                   0.015544      0.492252
min                   0.000259      0.000000
25%                   0.002787      0.000000
50%                   0.005432      0.000000
75%                   0.011786      1.000000
max                   0.178011      1.000000
```

Figure 2: Features in the Dataset

After the dataset merging, some continuous preprocessing steps were done to refine the data for model training. Same entries were removed to ensure data integrity, and irrelevant columns such as timestamp, user_id, and url_requested were dropped [28]. Data types were standardized, and missing values were addressed. Categorical variables, including device_type, region, and request_method, were encoded using label encoding.

***Article Received:** 25 March 2023 **Revised:** 12 April 2023 **Accepted:** 30 May 2023*

```
df = df.drop_duplicates()

df = df.drop(columns=["timestamp", "user_id", "url_requested"], axis=1)

df["response_size"] = pd1.to_numeric(df["response_size"], errors="coerce")

df["time_taken_ms"] = pd1.to_numeric(df["time_taken_ms"], errors="coerce")

df["request_size_time_ratio"]=pd1.to_numeric(df["request_size_time_ratio"],
errors="coerce")

for col in ["device_type", "region", "request_method"]:

  le1 = LabelEncoder()

  df[col] = le1.fit_transform(df[col].astype(str))

df["response_size"] = df["response_size"] / df["response_size"].max()

df["time_taken_ms"] = df["time_taken_ms"] / df["time_taken_ms"].max()

df["request_size_time_ratio"]=df["request_size_time_ratio"]df["request_size_time_ra
tio"].max()

df.to_csv("full_data_processed.csv", index=False)
```

Feature importance scores showed the following top contributors:

request_count
request_size_time_ratio
response_size
device_type

These insights helped understand which request attributes were most important f o r influence caching decisions.

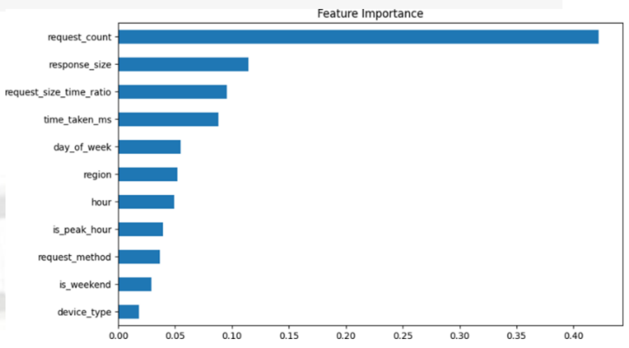


Figure .3: Feature Importance Analysis

Two models were trained using supervised classification [29][30]:

Random Forest Classifier
XGBoost Classifier

All algorithms had trained using 80% part of the dataset, with the remaining 20% part kept aside to evaluate its performance on unseen data.Hyperparameters were initially set using defaults;

***Article Received:** 25 March 2023 **Revised:** 12 April 2023 **Accepted:** 30 May 2023*

tuning may be extended via GridSearchCV

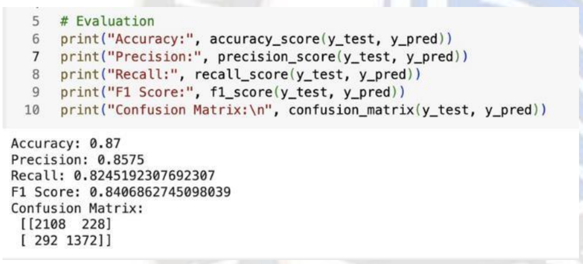
```
from sklearn.ensemble import RandomForestClassifier

my_model = RandomForestClassifier(random_state=42)

my_model.fit(X_train_mkdt, y_train_mkd)
```

```
from xgboost import XGBClassifier

xgb_model = XGBClassifier(random_state=42)

xgb_model.fit(X_train_mkd, y_train_mkd)
```

4.1 Evaluation Metrics and Result

Evaluation focused on accuracy, calculated using .score() and accuracy_score()

functions. The results are as follows:



```
# Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.87
Precision: 0.8575
Recall: 0.8245192307692307
F1 Score: 0.8406862745098039
Confusion Matrix:
 [[2108  228]
 [ 292 1372]]
```

Figure 4: Evaluation Metrics

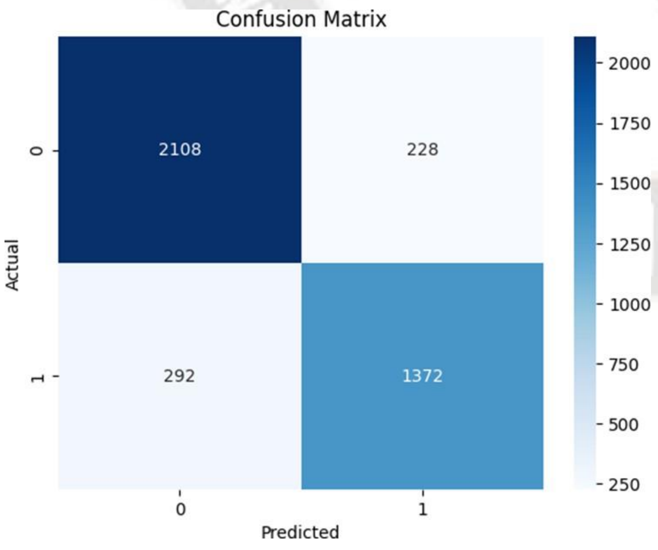


Figure 5: Confusion Matrix

4.2 Real-Time Caching Simulation

Redis was used as the caching backend. Based on the model's binary prediction (cache or not), a sample request was stored in Redis using a unique URL-based key [31].

Redis entries were monitored using the Streamlit interface, showing whether a request was cached along with associated metadata (timestamp, region, device type, etc.).

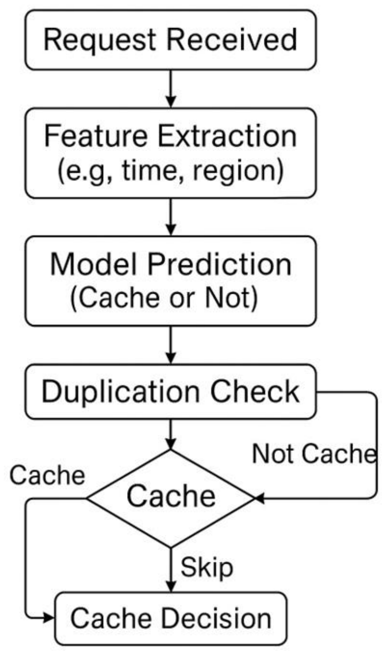


Figure 6: Prediction Process

***Article Received:** 25 March 2023 **Revised:** 12 April 2023 **Accepted:** 30 May 2023*

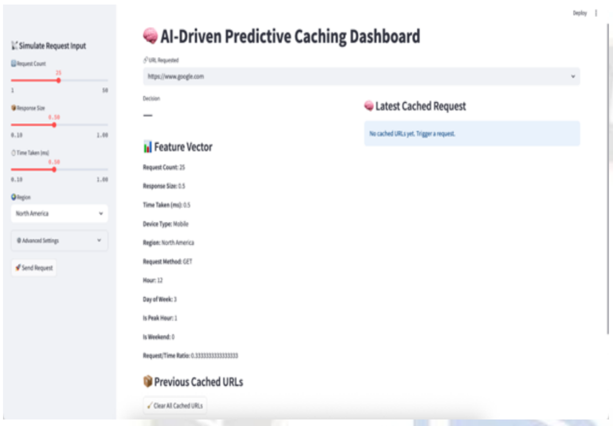
4.3 Streamlit App

The dashboard has been made using Streamlit to allow real-time simulation:

Input of randomized or user-defined request data

Visualization of prediction results (caching decision)

This served as a user interface to validate system behavior and showcase edge-node-like operations [32].



Simulate Request Input
Request Count: 25 (1–50)
Response Size: 0.50 (0.10–1.00)
Time Taken (ms): 0.50 (0.10–1.00)
Region: North America
Advanced Settings
Send Request

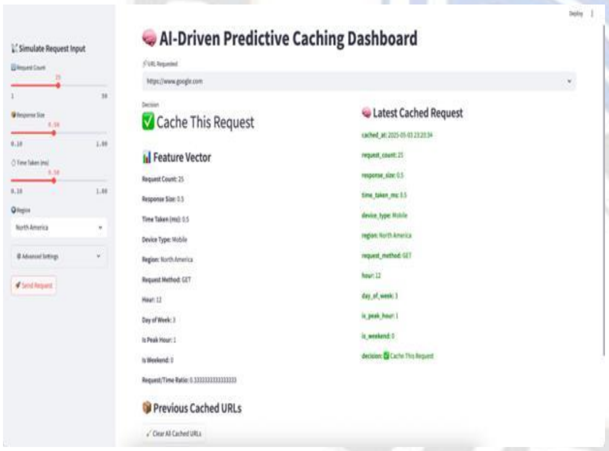
AI-Driven Predictive Caching Dashboard
URL Requested: https://www.google.com
Decision: —

Feature Vector
Request Count: 25
Response Size: 0.5
Time Taken (ms): 0.5
Device Type: Mobile
Region: North America
Request Method: GET
Hour: 12
Day of Week: 3
Is Peak Hour: 1
Is Weekend: 0
Request/Time Ratio: 0.3333333333333333

Latest Cached Request
No cached URLs yet. Trigger a request.

Previous Cached URLs
Clear All Cached URLs

Figure 7 (a) Dashboard UI showcasing overall system interface and analytics panel



Simulate Request Input
Request Count: 25 (1–50)
Response Size: 0.50 (0.10–1.00)
Time Taken (ms): 0.50 (0.10–1.00)
Region: North America
Advanced Settings
Send Request

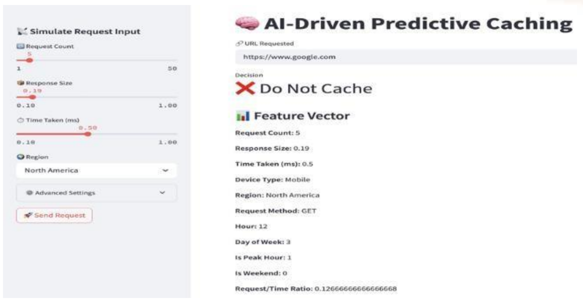
AI-Driven Predictive Caching Dashboard
URL Requested: https://www.google.com
Decision: ✅ Cache This Request

Feature Vector
Request Count: 25
Response Size: 0.5
Time Taken (ms): 0.5
Device Type: Mobile
Region: North America
Request Method: GET
Hour: 12
Day of Week: 3
Is Peak Hour: 1
Is Weekend: 0
Request/Time Ratio: 0.3333333333333333

Latest Cached Request
cached_at: 2025-05-03 23:20:34
request_count: 25
response_size: 0.5
time_taken_ms: 0.5
device_type: Mobile
region: North America
request_method: GET
hour: 12
day_of_week: 3
is_peak_hour: 1
is_weekend: 0
decision: ✅ Cache This Request

Previous Cached URLs
Clear All Cached URLs

Figure 7 (b) Visualization of a request being cached by the system

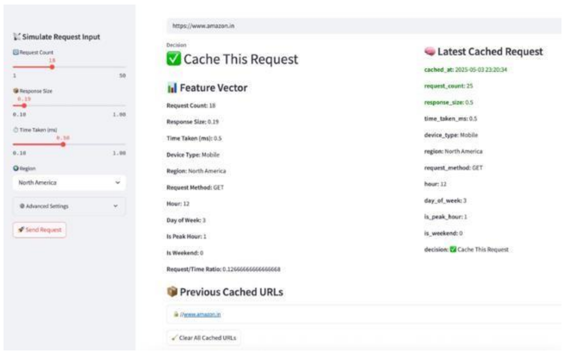


Simulate Request Input
Request Count: 5 (1–50)
Response Size: 0.19 (0.10–1.00)
Time Taken (ms): 0.50 (0.10–1.00)
Region: North America
Advanced Settings
Send Request

AI-Driven Predictive Caching
URL Requested: https://www.google.com
Decision: ❌ Do Not Cache

Feature Vector
Request Count: 5
Response Size: 0.19
Time Taken (ms): 0.5
Device Type: Mobile
Region: North America
Request Method: GET
Hour: 12
Day of Week: 3
Is Peak Hour: 1
Is Weekend: 0
Request/Time Ratio: 0.12666666666666668

Figure 7 (c) Example of a request identified as non-cacheable.



Simulate Request Input
Request Count: 18 (1–50)
Response Size: 0.19 (0.10–1.00)
Time Taken (ms): 0.50 (0.10–1.00)
Region: North America
Advanced Settings
Send Request

https://www.amazon.in
Decision: ✅ Cache This Request

Feature Vector
Request Count: 18
Response Size: 0.19
Time Taken (ms): 0.5
Device Type: Mobile
Region: North America
Request Method: GET
Hour: 12
Day of Week: 3
Is Peak Hour: 1
Is Weekend: 0
Request/Time Ratio: 0.12666666666666668

Latest Cached Request
cached_at: 2025-05-03 23:20:34
request_count: 25
response_size: 0.5
time_taken_ms: 0.5
device_type: Mobile
region: North America
request_method: GET
hour: 12
day_of_week: 3
is_peak_hour: 1
is_weekend: 0
decision: ✅ Cache This Request

Previous Cached URLs
//www.amazon.in
Clear All Cached URLs

Figure 7 (d) Display of previously accessed URLs, reflecting request history and system logging.

5. Conclusion

This work focused on designing and assessing an AI-based predictive caching system tailored for edge computing scenarios. Conventional caching techniques like LRU and LFU depend on fixed heuristics and fail to adapt to evolving user behavior patterns. To overcome these limitations, a Random Forest classifier was employed to determine the caching relevance of incoming requests. The model made decisions based on input features, such as timestamp, geographical region, device category, and frequency of requests

The system was trained on a synthetically generated dataset and integrated into a simulated edge architecture using Redis for caching and Streamlit for interactive visualization. The overall architecture successfully emulated real-time request prediction and caching logic, achieving an accuracy of 86.35% with balanced precision and recall. Feature importance analysis revealed that request count, the request size-to-time ratio, and response size were the most significant factors impacting the model's predictions.

This work shows the practical usage of deploying lightweight AI models in edge-like environments. By merging supervised machine learning with real-time simulation tools, the project highlights how caching decisions can be automated and optimized using data-driven techniques.

References

1. Goudarzi, M., & Hosseinzadeh, M. (2020). Deep Learning for Proactive Content Caching in Edge Networks. *Computer Networks*, 180, 107373.
2. Gao, L., Zhao, H., & Shen, M. (2021). Context-Aware Caching for Smart Cities Using Machine Learning. *IEEE Internet of Things Journal*, 8(6), 4839–4850.

3. Alkadi, M., Ahmed, A., & Mansour, A. (2022). Lightweight Machine Learning Models for IoT Edge Caching. *Sensors*, 22(1), 158.
4. Keshari, M., Verma, V., & Sharma, R. (2023). Deep Learning-Based Content Caching in CDN Systems: A CNN-GRU Hybrid Approach. *Journal of Network and Computer Applications*, 214, 103535.
5. Garg, P., Dixit, A., & Sethi, P. (2022). Ml-fresh: novel routing protocol in opportunistic networks using machine learning. *Computer Systems Science & Engineering, Forthcoming*. Tech Science Press.
6. Yadav, P. S., Khan, S., Singh, Y. V., Garg, P., & Singh, R. S. (2022). A Lightweight Deep Learning-Based Approach for Jazz Music Generation in MIDI Format. *Computational Intelligence and Neuroscience*, *2022*.
7. Soni, E., Nagpal, A., Garg, P., & Pinheiro, P. R. (2022). Assessment of Compressed and Decompressed ECG Databases for Telecardiology Applying a Convolution Neural Network. *Electronics*, *11*(17), 2708.
8. Pustokhina, I. V., Pustokhin, D. A., Lydia, E. L., Garg, P., Kadian, A., & Shankar, K. (2021). Hyperparameter search based convolution neural network with Bi-LSTM model for intrusion detection system in multimedia big data environment. *Multimedia Tools and Applications*, 1-18.
9. Khanna, A., Rani, P., Garg, P., Singh, P. K., & Khamparia, A. (2021). An Enhanced Crow Search Inspired Feature Selection Technique for Intrusion Detection Based Wireless Network System. *Wireless Personal Communications*, 1-18.
10. Garg, P., Dixit, A., Sethi, P., & Pinheiro, P. R. (2020). Impact of node density on the qos parameters of routing protocols in opportunistic networks for smart spaces. *Mobile Information Systems*, *2020*.
11. Beniwal, S., Saini, U., Garg, P., & Joon, R. K. (2021). Improving performance during camera surveillance by integration of edge detection in IoT system. *International Journal of E-Health and Medical Communications (IJEHMC)*, *12*(5), 84-96.
12. Garg, P., Dixit, A., & Sethi, P. (2019). Wireless sensor networks: an insight review. *International Journal of Advanced Science and Technology*, *28*(15), 612-627.
13. Sharma, N., & Garg, P. (2022). Ant colony based optimization model for QoS-Based task scheduling in cloud computing environment. *Measurement: Sensors*, 100531.
14. Kumar, P., Kumar, R., & Garg, P. (2020). Hybrid Crowd Cloud Routing Protocol For Wireless Sensor Networks.
15. Dixit, A., Garg, P., Sethi, P., & Singh, Y. (2020, April). TVCCCS: Television Viewer’s Channel Cost Calculation System On Per Second Usage. In *IOP Conference Series: Materials Science and Engineering* (Vol. 804, No. 1, p. 012046). IOP Publishing.
16. Dixit, A., Garg, P., Sethi, P., & Singh, Y. (2020, April). TVCCCS: Television Viewer’s Channel Cost Calculation System On Per Second Usage. In *IOP Conference Series: Materials Science and Engineering* (Vol. 804, No. 1, p. 012046). IOP Publishing.
17. Sethi, P., Garg, P., Dixit, A., & Singh, Y. (2020, April). Smart number cruncher–a voice based calculator. In *IOP Conference Series: Materials Science and Engineering* (Vol. 804, No. 1, p. 012041). IOP Publishing.
18. S. Rai, V. Choubey, Suryansh and P. Garg, "A Systematic Review of Encryption and Keylogging for Computer System Security," *2022 Fifth International Conference on Computational Intelligence and Communication Technologies (CCICT)*, 2022, pp. 157-163, doi: 10.1109/CCiCT56684.2022.00039.
19. L. Saraswat, L. Mohanty, P. Garg and S. Lamba, "Plant Disease Identification Using Plant Images," 2022 Fifth International Conference on Computational Intelligence and Communication Technologies (CCICT), 2022, pp. 79-82, doi: 10.1109/CCiCT56684.2022.00026.
20. L. Mohanty, L. Saraswat, P. Garg and S. Lamba, "Recommender Systems in E-Commerce," 2022 Fifth International Conference on Computational Intelligence and Communication Technologies (CCICT), 2022, pp. 114-119, doi: 10.1109/CCiCT56684.2022.00032.
21. C. Maggo and P. Garg, "From linguistic features to their extractions: Understanding the semantics of a concept," 2022 Fifth International Conference on Computational Intelligence and Communication Technologies (CCICT), 2022, pp. 427-431, doi: 10.1109/CCiCT56684.2022.00082.
22. N. Puri, P. Saggar, A. Kaur and P. Garg, "Application of ensemble Machine Learning models for phishing detection on web networks," 2022 Fifth International Conference on Computational Intelligence and Communication Technologies (CCICT), 2022, pp. 296-303, doi: 10.1109/CCiCT56684.2022.00062.

23. R. Sharma, S. Gupta and P. Garg, "Model for Predicting Cardiac Health using Deep Learning Classifier," 2022 Fifth International Conference on Computational Intelligence and Communication Technologies (CCICT), 2022, pp. 25-30, doi: 10.1109/CCiCT56684.2022.00017.
24. Varshney, S. Lamba and P. Garg, "A Comprehensive Survey on Event Analysis Using Deep Learning," 2022 Fifth International Conference on Computational Intelligence and Communication Technologies (CCICT), 2022, pp. 146-150, doi: 10.1109/CCiCT56684.2022.00037.
25. Dixit, A., Sethi, P., Garg, P., & Pruthi, J. (2022, December). Speech Difficulties and Clarification: A Systematic Review. In *2022 11th International Conference on System Modeling & Advancement in Research Trends (SMART)* (pp. 52-56). IEEE.
26. Chaudhary, A., & Garg, P. (2014). Detecting and diagnosing a disease by patient monitoring system. *International Journal of Mechanical Engineering And Information Technology*, *2*(6), 493-499.
27. Malik, K., Raheja, N., & Garg, P. (2011). Enhanced FP-growth algorithm. *International Journal of Computational Engineering and Management*, *12*, 54-56.
28. Garg, P., Dixit, A., & Sethi, P. (2021, May). Link Prediction Techniques for Opportunistic Networks using Machine Learning. In *Proceedings of the International Conference on Innovative Computing & Communication (ICICC)*.
29. Garg, P., Dixit, A., & Sethi, P. (2021, April). Opportunistic networks: Protocols, applications & simulation trends. In *Proceedings of the International Conference on Innovative Computing & Communication (ICICC)*.
30. Garg, P., Dixit, A., & Sethi, P. (2021). Performance comparison of fresh and spray & wait protocol through one simulator. *IT in Industry*, *9*(2).
31. Malik, M., Singh, Y., Garg, P., & Gupta, S. (2020). Deep Learning in Healthcare system. *International Journal of Grid and Distributed Computing*, *13*(2), 469-468.
32. Gupta, M., Garg, P., Gupta, S., & Joon, R. (2020). A Novel Approach for Malicious Node Detection in Cluster-Head Gateway Switching Routing in Mobile Ad Hoc Networks. *International Journal of Future Generation Communication and Networking*, *13*(4), 99-111.
33. Gupta, A., Garg, P., & Sonal, Y. S. (2020). Edge Detection Based 3D Biometric System for Security of Web-Based Payment and Task Management Application. *International Journal of Grid and Distributed Computing*, *13*(1), 2064-2076.
34. Kumar, P., Kumar, R., & Garg, P. (2020). Hybrid Crowd Cloud Routing Protocol For Wireless Sensor Networks.
35. Garg, P., & Raman, P. K. Broadcasting Protocol & Routing Characteristics With Wireless ad-hoc networks.
36. Garg, P., Arora, N., & Malik, T. Capacity Improvement of WI-MAX In presence of Different Codes WI-MAX: Speed & Scope of future.
37. Garg, P., Saroha, K., & Lochab, R. (2011). Review of wireless sensor networks-architecture and applications. IJCSMS International Journal of Computer Science & Management Studies, 11(01), 2231-5268.
38. Yadav, S., &Garg, P. Development of a New Secure Algorithm for Encryption and Decryption of Images.
39. Dixit, A., Sethi, P., & Garg, P. (2022). Rakshak: A Child Identification Software for Recognizing Missing Children Using Machine Learning-Based Speech Clarification. International Journal of Knowledge-Based Organizations (IJKBO), 12(3), 1-15.
40. Shukla, N., Garg, P., & Singh, M. (2022). MANET Proactive and Reactive Routing Protocols: A Comparison Study. International Journal of Knowledge-Based Organizations (IJKBO), 12(3), 1-14.
41. Chauhan, S., Singh, M., & Garg, P. (2021). Rapid Forecasting of Pandemic Outbreak Using Machine Learning. *Enabling Healthcare 4.0 for Pandemics: A Roadmap Using AI, Machine Learning, IoT and Cognitive Technologies*, 59-73.
42. Gupta, S., & Garg, P. (2021). An insight review on multimedia forensics technology. *Cyber Crime and Forensic Computing: Modern Principles, Practices, and Algorithms*, *11*, 27.
43. Shrivastava, P., Agarwal, P., Sharma, K., & Garg, P. (2021). Data leakage detection in Wi-Fi networks. *Cyber Crime and Forensic Computing: Modern Principles, Practices, and Algorithms*, *11*, 215.
44. Meenakshi, P. G., & Shrivastava, P. (2021). Machine learning for mobile malware analysis. *Cyber Crime and Forensic Computing: Modern Principles, Practices, and Algorithms*, *11*, 151.
45. Garg, P., Pranav, S., & Prerna, A. (2021). Green Internet of Things (G-IoT): A Solution for Sustainable Technological Development. In *Green*

Internet of Things for Smart Cities (pp. 23-46). CRC Press.

46. Nanwal, J., Garg, P., Sethi, P., & Dixit, A. (2021). Green IoT and Big Data: Succeeding towards Building Smart Cities. In *Green Internet of Things for Smart Cities* (pp. 83-98). CRC Press.
47. Gupta, M., Garg, P., & Agarwal, P. (2021). Ant Colony Optimization Technique in Soft Computational Data Research for NP-Hard Problems. In *Artificial Intelligence for a Sustainable Industry 4.0* (pp. 197-211). Springer, Cham.
48. Magoo, C., & Garg, P. (2021). Machine Learning Adversarial Attacks: A Survey Beyond. *Machine Learning Techniques and Analytics for Cloud Security*, 271-291.